Decision Procedures for Separation Logic

Alessio Mansutti

Barbizon 2018
Hoare calculus is based on proof rules manipulating Hoare triples.

\[\{P\} \ C \ \{Q\}\]

where

- \(C\) is a program
- \(P\) and \(Q\) are assertions in some logical language.

Any (memory) state that satisfies \(P\) will satisfy \(Q\) after being modified by \(C\).
Programming languages with pointers

The so-called **frame rule**

\[
\begin{array}{c}
\{P\} \ C \ \{Q\} \\
\{F \land P\} \ C \ \{F \land Q\}
\end{array}
\]

is generally not valid: it fails if \( C \) manipulates pointers.
Programming languages with pointers

The so-called **frame rule**

\[
\begin{align*}
\{P\} & \ C \ \{Q\} \\
\{F \land P\} & \ C \ \{F \land Q\}
\end{align*}
\]

is generally not valid: it fails if \(C\) manipulates pointers.

Example:

\[
\begin{align*}
\{\exists u. x \leftrightarrow u\} & \ [x] \leftarrow 4 \ \{x \leftrightarrow 4\} \\
\{y \leftrightarrow 3 \land \exists u. x \leftrightarrow u\} & \ [x] \leftarrow 4 \ \{y \leftrightarrow 3 \land x \leftrightarrow 4\}
\end{align*}
\]

not true if \(x\) and \(y\) are in aliasing.
Separation logic add the notion of \textit{separation} (\textasteriskcentered) of a state, so that the frame rule

\[
\{P\} \ C \ \{Q\} \ \text{modv}(C) \cap \text{fv}(F) = \emptyset
\]

\[
\{F \ast P\} \ C \ \{F \ast Q\}
\]

is valid.
Reynolds’02: Separation logic

Separation logic add the notion of separation (⋆) of a state, so that the frame rule

\[
\begin{align*}
\{P\} & \quad C \quad \{Q\} \quad \text{modv}(C) \cap \text{fv}(F) = \emptyset \\
\{F \star P\} & \quad C \quad \{F \star Q\}
\end{align*}
\]

is valid.

**Automatic Verifiers:** Infer, SLAyer, Predator (all 2011).

Why we need decision procedures for SL?

- Many tools support fragments of Separation Logic as an assertion language.

- Growing demand to consider more powerful extensions:
  - inductive predicates;
  - magic wand operator $\neg\ast$;
  - closure under boolean connectives.

- Deciding satisfiability/validity/entailment is needed.

$$P \implies P' \quad \{P'\} \ C \ \{Q'\} \quad Q' \implies Q$$

consequence rule
Memory states with one record field

Separation Logic is interpreted over memory states \((s, h)\) where:
- \(s : \text{VAR} \rightarrow \text{LOC}\) is called store;
- \(h : \text{LOC} \rightarrow_{\text{fin}} \text{LOC}\) is called heap.

where \(\text{VAR} = \{x, y, z, \ldots\}\) set of (program) variables;
\(\text{LOC}\) set of locations (typically \(\text{LOC} \cong \mathbb{N} \cong \text{VAR}\)).
Propositional Separation Logic $\text{SL}(\ast, \rightarrow)$

$\varphi := \neg \varphi \mid \varphi_1 \land \varphi_2 \mid x = y \mid \text{emp} \mid x \leftrightarrow y \mid \varphi_1 \ast \varphi_2 \mid \varphi_1 \rightarrow \varphi_2$

Semantics

- standard for $\land$ and $\neg$;
- $(s, h) \models x = y \iff s(x) = s(y)$
- $(s, h) \models \text{emp} \iff \text{dom}(h) = \emptyset$
- $(s, h) \models x \leftrightarrow y \iff h(s(x)) = s(y)$
separating conjunction (∗)

\((s, h) \models \varphi_1 \ast \varphi_2\) if and only if

\[(s, h_2) \models \varphi_2\]

\[(s, h) \models \varphi_1\]

There is a way to split the heap into two so that, together with the store, one part satisfies \(\varphi_1\) and the other satisfies \(\varphi_2\).
Separating implication ($\rightarrow \star$)

$$(s, h) \models \varphi_1 \rightarrow \varphi_2 \text{ if and only if }$$

$$\forall h_1 \quad \text{dom}(h) \cap \text{dom}(h_1) = \emptyset$$

$$(s, h_1) \models \varphi_1$$

$$(s, h + h_1) \models \varphi_2$$

Whenever a (disjoint) heap that, together with the store, satisfies $\varphi_1$ is added, the resulting memory state satisfies $\varphi_2$. 
Symbolic Heap Fragment (SHF)

\[ \Sigma := \text{emp} \mid x \mapsto y \mid \text{ls}(x, y) \mid \Sigma \ast \Sigma \]
\[ \Pi := x = y \mid x \neq y \mid \Pi \land \Pi \]
\[ \phi := \Sigma \land \Pi \]

- standard fragment in automated tools;
- satisfiability/entailment in \( \text{PTime} \);
- boolean combination of SHF is \( \text{NP}-\text{complete} \);
Extension: \( \text{SL}(\ast, \neg\ast) + \text{list segment predicate (ls)} \)

\[(s, h) \models \text{ls}(x, y) \text{ if and only if }\]

\[s(x) \rightarrow \bullet \rightarrow \bullet \rightarrow \bullet \rightarrow \bullet \rightarrow \bullet \rightarrow s(y)\]

\(s(x)\) reaches \(s(y)\) and all elements in \(\text{dom}(h)\) are necessary for this to hold.

Note: \(\text{SL}(\ast, \neg\ast)\) is already \(\text{PSPACE}\)-complete.
The satisfiability problem for $\text{SL}(\ast, \neg\ast, 1\text{s})$ is undecidable.

Several variants of $\text{SL}(\ast, \neg\ast, 1\text{s})$ are also concluded undecidable.

The satisfiability problem for $\text{SL}(\ast, 1\text{s})$ (i.e. $\text{SL}(\ast, \neg\ast, 1\text{s})$ without $\neg\ast$) is $\text{PSPACE}$-complete.

The satisfiability problem for Boolean combinations of formulae in $\text{SL}(\ast, 1\text{s}) \cup \text{SL}(\ast, \neg\ast)$ is $\text{PSPACE}$-complete.
Undecidability of SL(\(*, \neg\)*)

As soon as we add to SL(\(*, \neg\)*) predicates so that it can express

- alloc\(^{-1}\)(x) ⇔ • → s(x)
- \(n(x) = n(y)\) ⇔ s(x) → • ← s(y)
- \(n(x) \leftrightarrow n(y)\) ⇔ s(x) → • ← • ← s(y)

we obtain a logic with undecidable satisfiability problem.

For example:

- SL(\(*, \neg\)*) + reach(x, y) = 2 and reach(x, y) = 3;
- SL(\(*, \neg\), 1s).
Reduction of First-order SL(\(\neg \ast\)) to SL(\(\ast, \neg \ast, 1s\))

- We consider the first-order extension of SL(\(\neg \ast\))

\[(s, h) \models \forall x. \varphi \iff \text{for all } \ell \in \text{LOC}, (s[x \leftarrow \ell], h) \models \varphi\]

- The satisfiability problem for First-order SL(\(\neg \ast\)) is undecidable. [IC, 2012].

- Idea for the translation: use the heap to mimic the store.
Heaps simulate stores

- Given $V \subseteq \text{fin VAR}$, take $s|_V + h : \text{VAR} + \text{LOC} \to \text{fin LOC}$ and translate it inside the heap domain $[\text{LOC} \to \text{fin LOC}]$;

- A finite set of locations is used to simulate a finite portion of the store, effectively splitting the domain LOC.
Undecidability – Some bits of the translation

- $\text{translation}_V(x = y) \overset{\text{def}}{=} n(x) = n(y)$;
- $\text{translation}_V(x \leftarrow y) \overset{\text{def}}{=} n(x) \leftarrow n(y)$;
- $\text{translation}_V(\varphi_1 \ast \varphi_2) \overset{\text{def}}{=} \text{too long for a slide}$;

Universal quantifier – $\forall x. \varphi$

$$(\text{alloc}(x) \land \text{size} = 1) \ast (\text{safe}(V) \implies \text{translation}_V(\varphi))$$

Where $\text{safe}(V)$ states the sanity conditions to encode the store.
Undecidability – Some bits of the translation

- translation$_V$(x = y) $\overset{\text{def}}{=} n(x) = n(y)$;
- translation$_V$(x $\hookrightarrow$ y) $\overset{\text{def}}{=} n(x) \hookrightarrow n(y)$;
- translation$_V$(ϕ$_1 \rightarrow$ ϕ$_2$) $\overset{\text{def}}{=} \text{too long for a slide}$;

Universal quantifier – $\forall x. \varphi$

$(\text{alloc}(x) \land \text{size} = 1) \rightarrow (\text{safe}(V) \implies \text{translation}_V(\varphi))$

Where safe$(V)$ states the sanity conditions to encode the store.
Deciding $SL(\ast, 1s)$ thanks to the test formulae approach

- Define sets $\text{Test}_X(n)$ that internalise the role of $\ast$;
- $\ast$ elimination: show that each formula of $SL(\ast, 1s)$ is captured by a boolean combination of test formulae;
- Show a small-model property for the logic of test formulae.

**Open problem**: to generalise this approach
- identify sufficient conditions on test formulae to have $\ast$ elimination;
- handle multiple families of test formulae;
* elimination (winning strategy for Duplicator)

For every

- \((s, h) \approx_n (s', h')\);
- \(n_1, n_2 \in \mathbb{N}^+\) such that \(n = n_1 + n_2\);
- \(h_1, h_2\) disjoint heaps such that \(h_1 + h_2 = h\) there are two disjoint heaps \(h'_1\) and \(h'_2\) such that
  - \(h'_1 + h'_2 = h'\);
  - \((s, h_1) \approx_{n_1} (s', h'_1)\) and \((s, h_2) \approx_{n_2} (s', h'_2)\).
Toy Test Formulae Test $\chi(n)$

- $(s, h) \models \#\text{loops}(\beta) \geq \beta' \iff$ the number of loops of size $\beta \leq \mathcal{G}(n)$ is at least $\beta'$;

- $(s, h) \models \#\text{loops}^{\uparrow} \geq \beta' \iff$ there are at least $\beta'$ loops of size at least $\mathcal{G}(n) + 1$;

- $(s, h) \models \text{garbage} \geq \beta \iff$ the number of locations not in a loop is at least $\beta$

where $\beta \in [1, \mathcal{G}(n)]$ and $\beta' \in [1, \mathcal{L}(n)]$.

Note: these formulae induce a partition on $h$. 
**elimination**

Let \((s, h) \approx_n (s', h')\) and let \(n_1, n_2 \in \mathbb{N}^+\) such that \(n = n_1 + n_2\).
For every \(h_1, h_2\) disjoint heaps such that \(h_1 + h_2 = h\)...

**Bound on garbage \(\geq \beta\) formulae**

Given \(h = h_1 + h_2\), every location not in a loop of \(h\) cannot be in a loop in \(h_1\) or \(h_2\). Then the bound \(G(n)\) must satisfy

\[
G(n) \geq \max_{n_1, n_2 \in \mathbb{N}^+, n_1 + n_2 = n} (G(n_1) + G(n_2))
\]
Bound on $\#\text{loops}$ formulae

We consider $\#\text{loops}(2) \geq \beta'$ (other cases are similar).

Take $h = h_1 + h_2$. Given a loop of size 2 in $h$, we identify three cases

- both locations of the loop are assigned to $h_1$;
- both locations of the loop are assigned to $h_2$;
- one location of the loop is assigned to $h_1$ and the other is assigned to $h_2$.

Then, we search for a bound $\mathcal{L}(n)$ on $\beta'$ such that

$$\mathcal{L}(n) \geq \max_{n_1, n_2 \in \mathbb{N}^+} \left( \mathcal{L}(n_1) + \mathcal{L}(n_2) + G(n_1) + G(n_2) \right)$$

subject to $n_1 + n_2 = n$. 

We have the inequalities

\[ G(1) \geq 1 \quad G(n) \geq \max_{n_1, n_2 \in \mathbb{N}^+, n_1 + n_2 = n} (G(n_1) + G(n_2)) \]

\[ L(1) \geq 1 \quad L(n) \geq \max_{n_1, n_2 \in \mathbb{N}^+, n_1 + n_2 = n} (L(n_1) + L(n_2) + G(n_1) + G(n_2)) \]

Which admit \( G(n) = n \) and \( L(n) = \frac{1}{2}n(n + 3) - 1 \) as a solution.

For the family \( \text{Test}_X(n) \)

\[
\begin{cases}
\#	ext{loops}(\beta) \geq \beta', \ \#\text{loops}^\uparrow \geq \beta', \\
\text{garbage} \geq \beta \\
\end{cases}
\quad \begin{array}{c}
\beta \in [1, n] \\
\beta' \in \left[1, \frac{1}{2}n(n + 3) - 1\right]
\end{array}
\]

we have * elimination.
Test formulae approach (after ∗ elimination)

Suppose we have a family of test formulae $\text{Test}_X(n)$, for all $n \in \mathbb{N}$, such that
- captures the atomic predicates of $\text{SL}(\ast, 1s)$;
- satisfies the $\ast$ elimination lemma.

Then, let $n \geq |\varphi|$ and $\text{var}(\varphi) \subseteq X$.
- If $(s, h) \approx_n (s', h')$ then we have $(s, h) \models \varphi$ iff $(s', h') \models \varphi$.
- $\varphi$ is logically equivalent to a Boolean combination of test formulae from $\text{Test}_X(n)$.

Small model property for boolean combination of $\text{Test}_X(n)$ formulae implies small model property for $\text{SL}(\ast, 1s)$. 
Extending FOSSACS paper: 1SL(*, ¬*, 1s)

SL(*, ¬*, 1s) with one quantified variable $u$, i.e.

$$(s, h) \models \forall u. \varphi \iff \text{for all } \ell \in \text{LOC}, (s[u \leftarrow \ell], h) \models \varphi$$

Has $\text{PSPACE}$-complete satisfiability problem when $1s(x, y)$ is constrained so that

- it does not occur on the right side of $\text{¬}$;
- if $x = u$ then also $y = u$.

Without the first condition: undecidable.
Without the second condition: $\text{TOWER}$-hard.

Proof using two families of test formulae.
Two families of test formulae

$$\Omega := \ldots \mid \exists u.\Omega \mid \Omega_1 \ast \Omega_2 \mid \Pi \rightarrow \ast \Omega$$

$$\Pi := \ldots \mid \text{reach}^+(x, e) \mid \text{reach}^+(u, u) \mid \exists u.\Pi \mid \Pi_1 \ast \Pi_2 \mid \Pi \rightarrow \ast \Omega$$

- Separately define test formulae for $\Omega$ and $\Pi$;
- $\ast$ elimination and quantifier elimination for both $\Omega$ and $\Pi$;
- Show that test formulae of $\Pi$ can express test formulae of $\Omega$. Then, prove $\rightarrow \ast$ elimination.
- Show small-model property for the logic of test formulae for $\Pi$. 
Fragment of $1SL(\ast, \neg\ast, 1s)$

- It subsumes other $\text{PSPACE}$-complete fragments of Separation Logic known in the literature;

- Weakening one of the two conditions most likely makes the problem escape $\text{PSPACE}$.

Also, first $\text{PSPACE}$ fragment of Separation Logic that can check

- garbage freedom: every model satisfying $\varphi$ has every memory cell reachable from a program variable occurring in $\varphi$.

- acycliclity: every model satisfying $\varphi$ is without loops.
Ongoing work

- Generalising the Test Formulae approach.

Logics

- SPIN’14: Existential fragment of Separation Logic;
- Separation Logic with Inductive Predicates or data values.

Verification

- (Bi-)abduction / Concurrency for SL with reachability;
- IJCAR’18: Fragment of SL with data values in SMT solver;